# 题目

给定一个包含 0, 1, 2, ..., n 中 n 个数的序列，找出 0 .. n 中没有出现在序列中的那个数。

**示例 1:**

输入: [3,0,1]

输出: 2

**示例 2:**

输入: [9,6,4,2,3,5,7,0,1]

输出: 8

**说明:**

你的算法应具有线性时间复杂度。你能否仅使用额外常数空间来实现?

# 分析

## 方法一：排序

class Solution {

public:

int missingNumber(vector<int>& nums) {

std::sort(nums.begin(),nums.end());

for(int i=0;i<nums.size();i++)

{

if(i != nums.at(i))

return i;

}

return nums.size();

}

};

或：

class Solution {

public:

int missingNumber(vector<int>& nums) {

sort(nums.begin(),nums.end());

int left = 0, right = nums.size(), mid= (left + right)/2;

while(left < right)

{

mid = (left + right)/2;

if(nums[mid] > mid) right = mid; //二分查找

else left = mid+1;

}

return left;

}

};

## 方法二：哈希表

**思路：**

我们可以直接查询每个数是否在数组中出现过来找出缺失的数字。如果使用哈希表，那么每一次查询操作都是常数时间的。

**代码：**

class Solution {

public:

int missingNumber(vector<int>& nums) {

unordered\_map<int, int> mp;

int ans;

for(int n : nums) mp[n] ++;

for(int i = 0; i <= nums.size(); i++){

if(mp[i] == 0){

ans = i;

break;

}

}

return ans;

}

};

**复杂度分析：**

时间复杂度：O(n)。集合的插入操作的时间复杂度都是O(1)，一共插入了n个数，时间复杂度为O(n)。集合的查询操作的时间复杂度同样是O(1)，最多查询n+1次，时间复杂度为O(n)。因此总的时间复杂度为O(n)。

空间复杂度：O(n)。集合中会存储n个数，因此空间复杂度为O(n)。

## 方法三：位运算

**思路：**

由于异或运算（XOR）满足结合律，并且对一个数进行两次完全相同的异或运算会得到原来的数，因此我们可以通过异或运算找到缺失的数字。

算法：

我们知道数组中有n个数，并且缺失的数在[0..n]中。因此我们可以先得到 [0..n]的异或值，再将结果对数组中的每一个数进行一次异或运算。未缺失的数在[0..n]和数组中各出现一次，因此异或后得到0。而缺失的数字只在[0..n]中出现了一次，在数组中没有出现，因此最终的异或结果即为这个缺失的数字。

在编写代码时，由于[0..n]恰好是这个数组的下标加上n，因此可以用一次循环完成所有的异或运算，例如下面这个例子：

下标 0 1 2 3

数字 0 1 3 4

可以将结果的初始值设为n，再对数组中的每一个数以及它的下标进行一个异或运算，即：

missing=4∧(0∧0)∧(1∧1)∧(2∧3)∧(3∧4)

=(4∧4)∧(0∧0)∧(1∧1)∧(3∧3)∧2

=0∧0∧0∧0∧2

=2

就得到了缺失的数字为 2。

**代码：**

class Solution {

public:

int missingNumber(vector<int>& nums) {

int res = nums.size();

for(int i = 0; i < nums.size(); ++i)

res = res ^ i ^ nums[i]; // a^b^b = a;

return res ;

}

};

**复杂度分析：**

时间复杂度：O(n)。这里假设异或运算的时间复杂度是常数的，总共会进行 O(n)次异或运算，因此总的时间复杂度为O(n)。

空间复杂度：O(1)。算法中只用到了O(1)的额外空间，用来存储答案。